Iterative Evaluation-Design Development

[Preece, Chap 19-22; Hix, Chap 4-5; Nielsen, Chap 4]

The Process of Interactive Software Development

Principles For the Process of User Interaction Development and its Management (Hix & Hartson)

- Development should include early and continous empirical testing, centered around appropriate users performing representative tasks.
- As development proceeds, it should incorporate subsequent iterative refinement procedures and cost/benefit analyses to determine the most cost effective changes to make to the user interaction design.
- The management process should verify and control the overall development life cycle and assign accountability for each step.

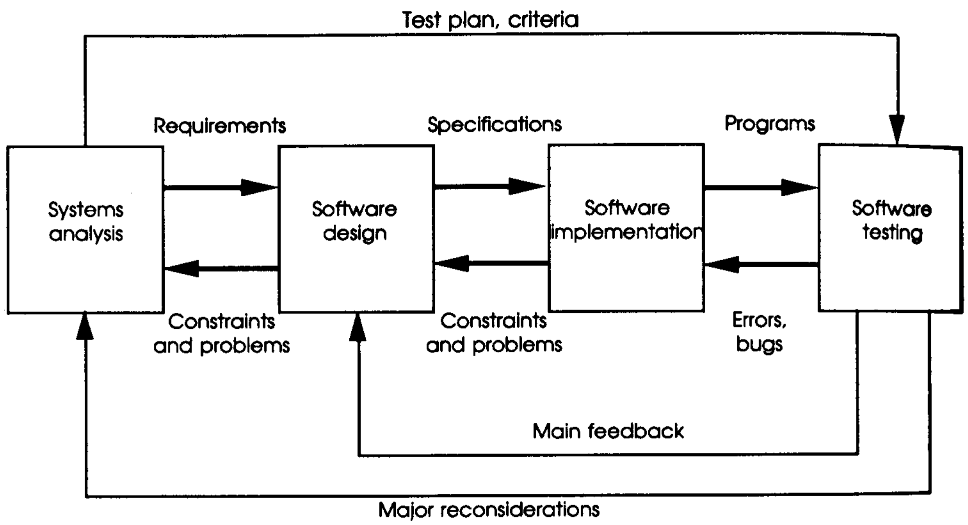


Figure 23.

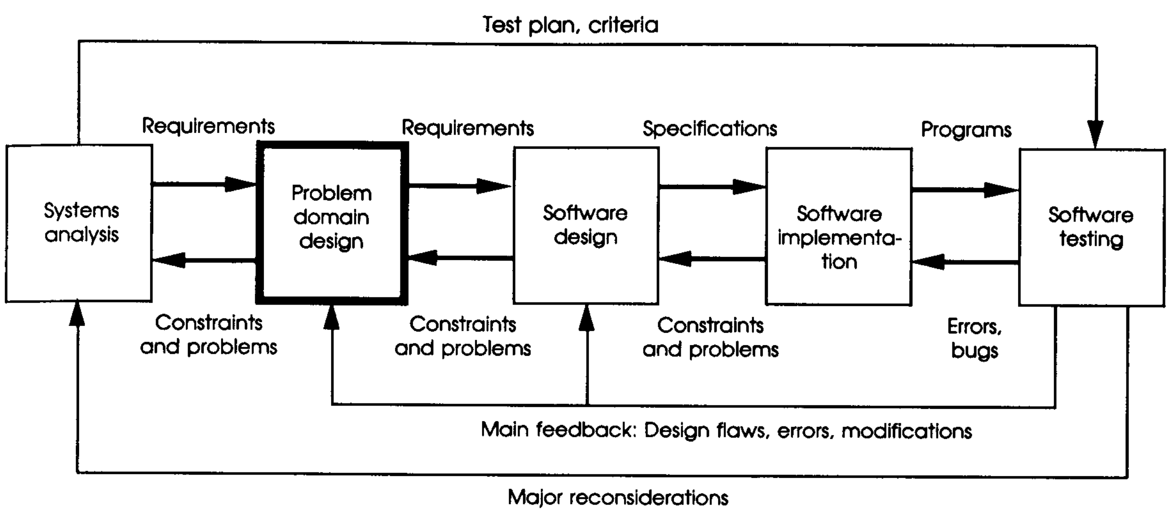


Figure 24.

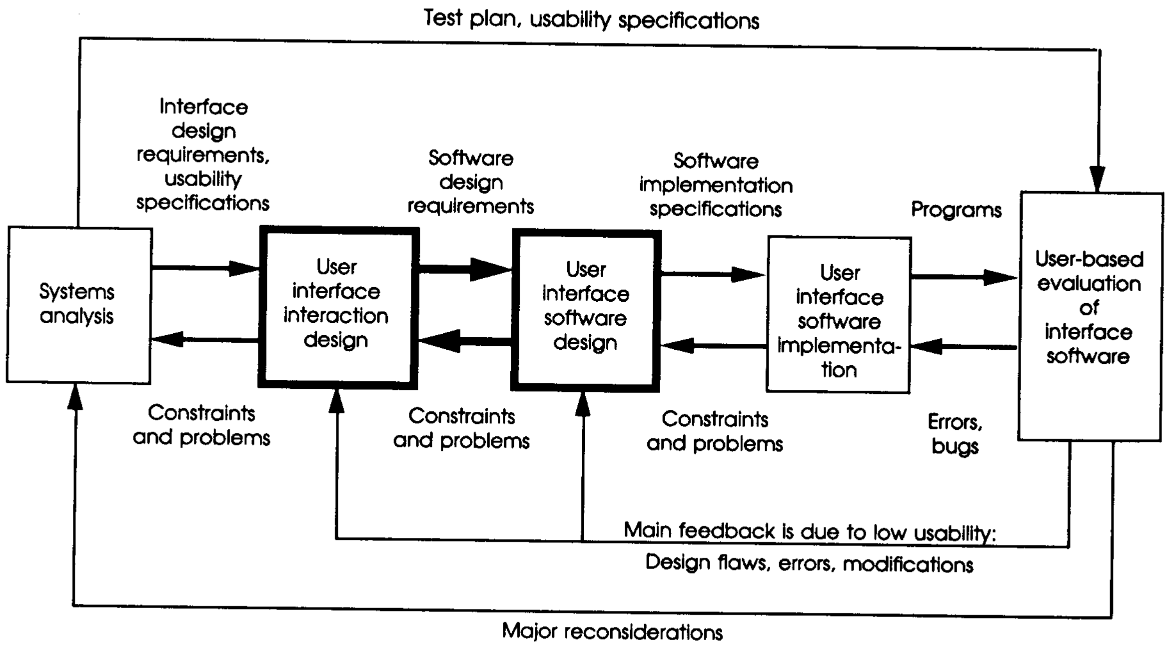


Figure 25.

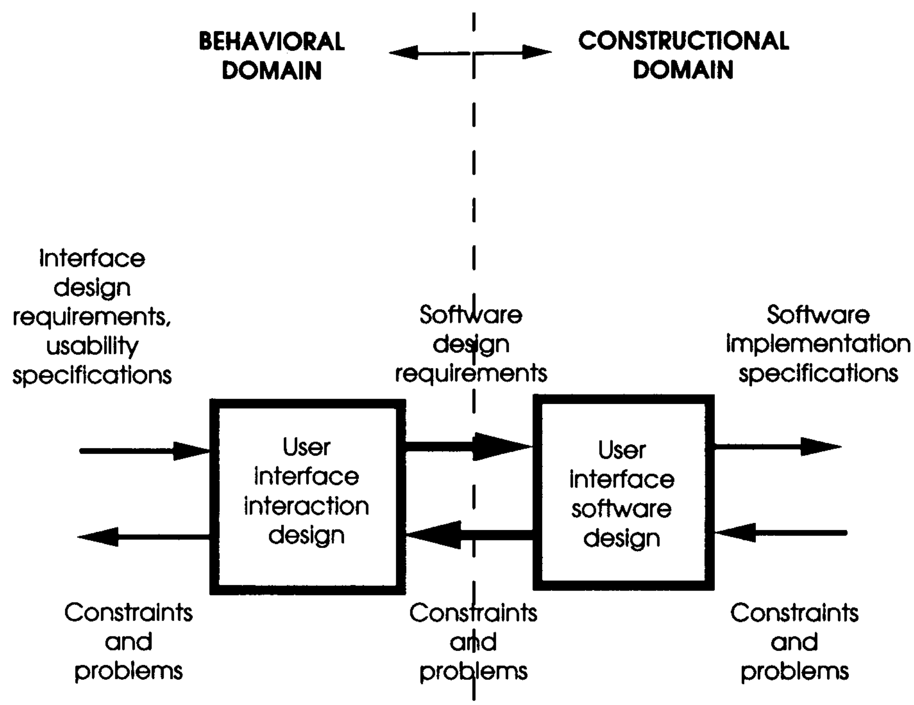


Figure 26.

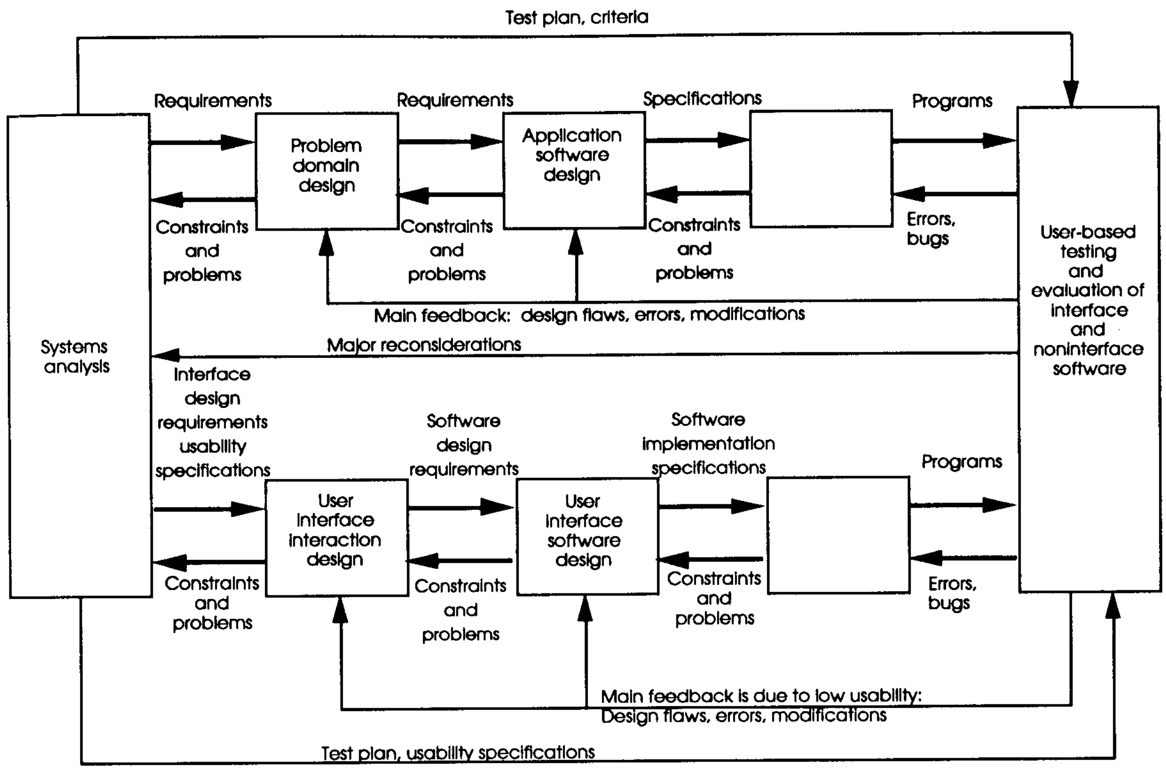


Figure 27.

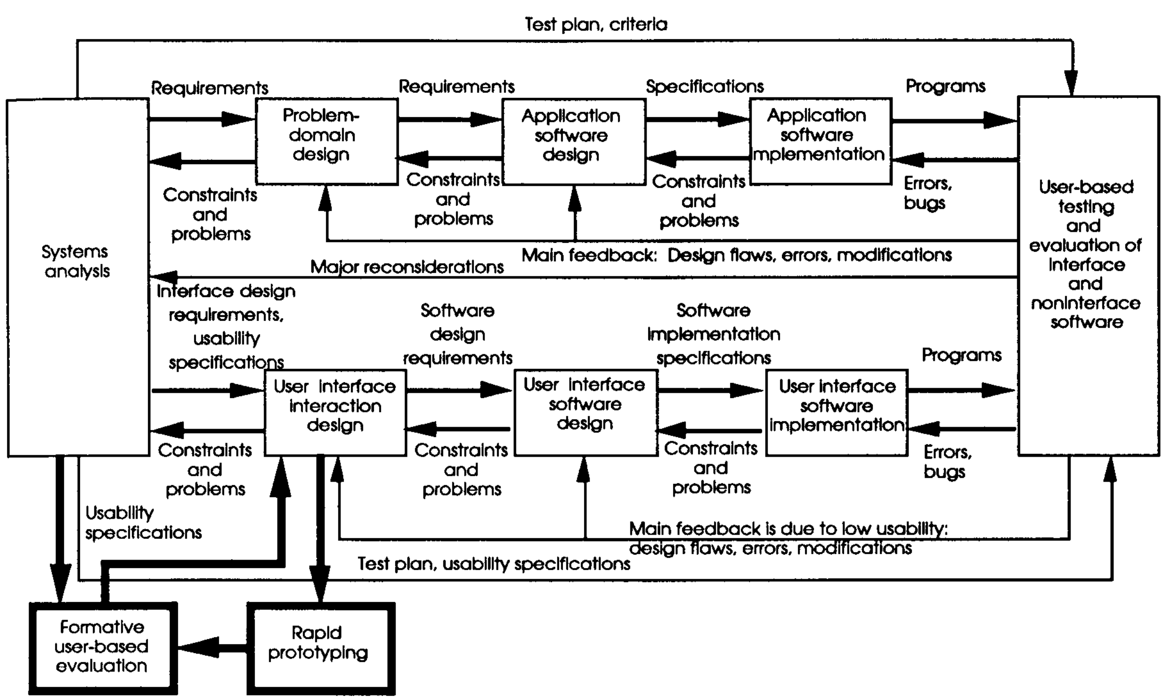


Figure 28.

What is being designed?
Two languages:
- user -> computer
- computer -> user

A communication protocol

What is the product of the design?
- A complete design document and/or prototype for all aspects of the user-computer interface
- Use formal definition tools and prototyping

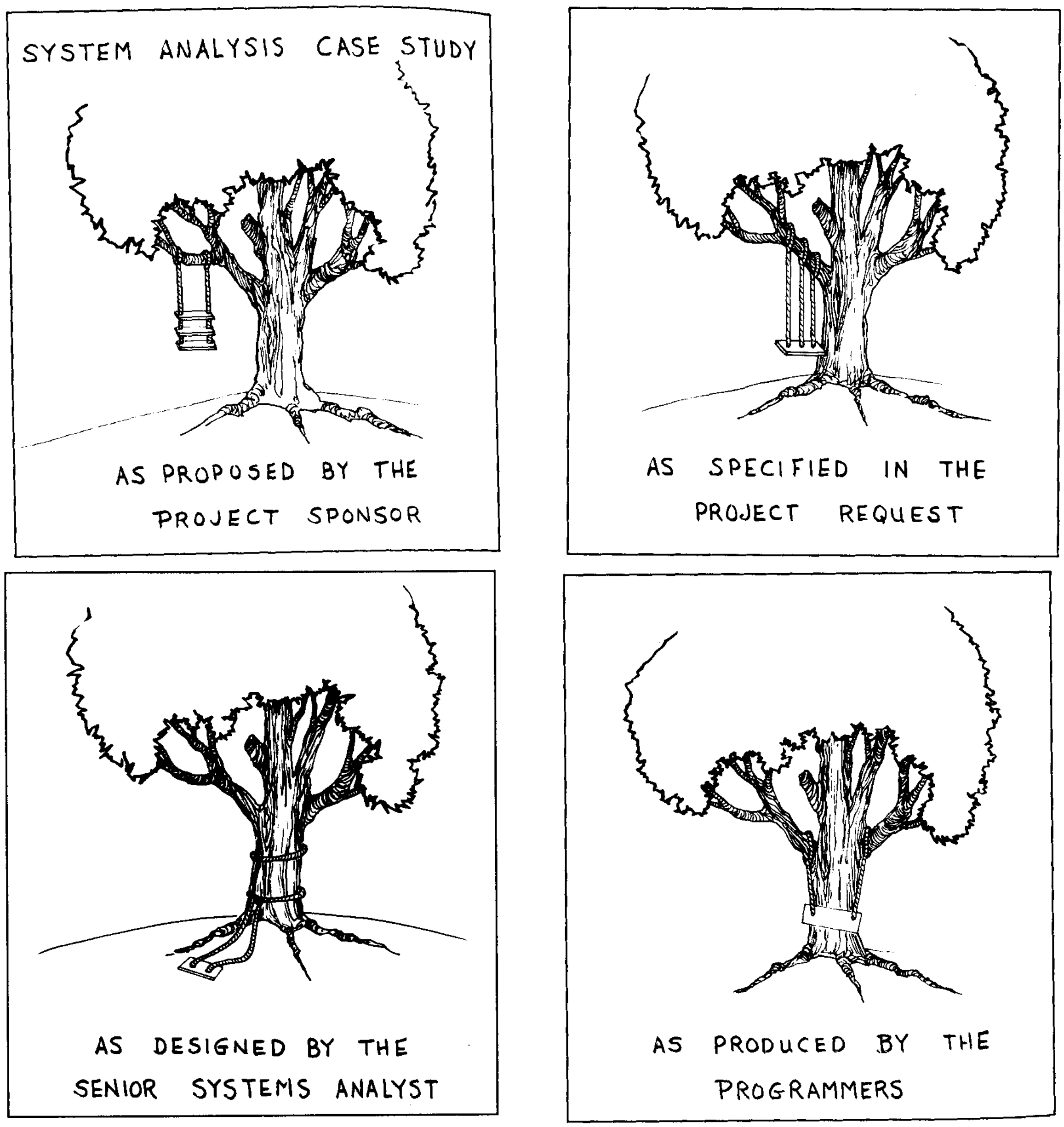
Requirements gathering

Composed of several parts:
- *Needs analysis*: This establishes that a new system is needed, based on the goals of the organization and the damands of the user base. This step determines the basic goals, purposes, and features desired for the application. The result is an external view of what users will be able to do with the system.
- *User analysis*: This combines cognitive theory of human users, specific information about job functions and tasks of potential users, plus social and organizational workflow considerations, to define representative classes of users in terms of the tasks to be performed and the skills and knowledge those users bring to the tasks. The result is a set of user class definitions, also called “user profiles”.
- *Task analysis*: This provides a complete description of the tasks, subtasks, and methods involved in using the new system, identifying resources necessary for users and the system cooperatively to perform these tasks. Task analysis usually results in a top-down decomposition of detailed task descriptions.
- *Functional analysis*: Similar to task analysis, this results in an internal view of the technical functions to be designed into the computational (noninterface) component of the system.
- Task/Function allocation: This results in decisions about what parts of the tasks will be performed by the user and what will be performed by the system. A distinction is made between manual and automated tasks.
- *Requirements analysis*: This is the formal process of specifying design requirements for the system. Requirements analysis draws on needs analysis, task analysis, and functional analysis to set the formal requirements.

Basic Principles
- Determine which parts are best done by computer, <u>but not how</u>
- “Know thy user” — total immersion
- Ask and observe
- People skills are crucial
- Difficult for technologists to see from user’s perspective

Results of requirements gathering
Design objectives
- Functional requirements
- Usability requirements
    - Learning time
    - Speed of use
    - Error rates
    - User satisfaction

Design constraints
- Existing equipment
- Compatability with other computer systems
- Implementation time and resources



User characteristics

- Personality
- Knowledge
- Work environment
- Morale
- Adaptability to change

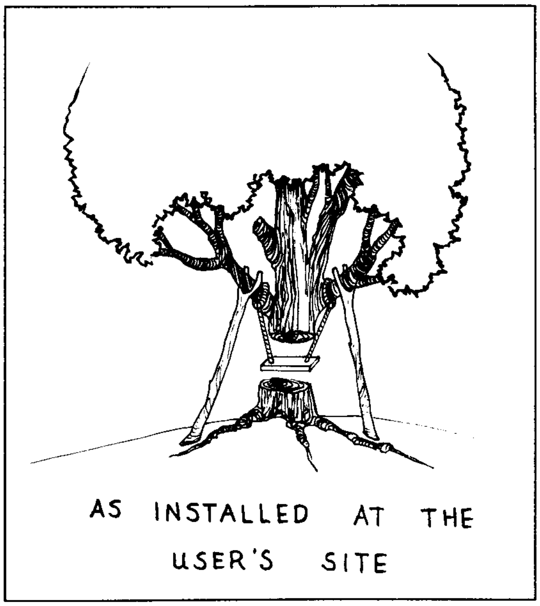
Training approaches

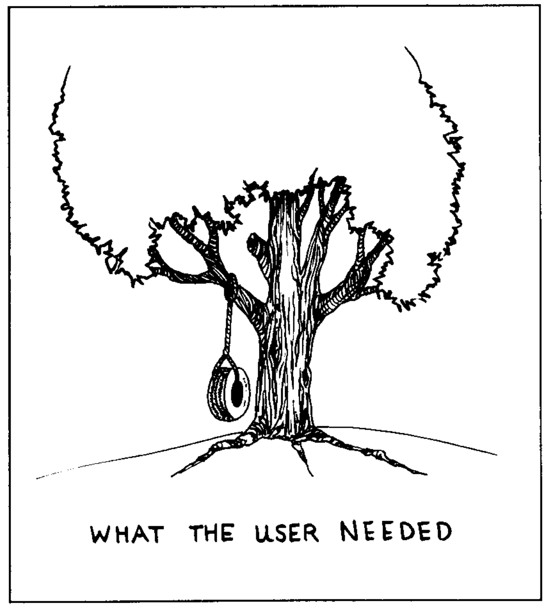
- On the job (embedded)
- Formal

User definition and perspective

Demographics

- age
- education





- cultural characteristics

Intrinsic personality factors
- attitude towards computers
- secure/insecure
- bold/timid
- adaptable/rigid
- motivated apathetic

Knowledge
- previous computer experience
- skill level (novice, intermediate, expert)
- intelligence
- reading ability
- typing ability

Work Environment
- frequency of computer use
- time allotted to learn system
- mental workload or overload
- stress level

Conceptual design

Identify key concepts in application:
- types of objects
- relations between objects
- attributes of objects
- actions on objects, relations, attributes

Identify a real world model or metaphor, if any:
- example: Mac desktop
- Use metaphor only if and when it is appropriate

Conceptual design example: a simple drawing system

Objects: page, line, point
Relations: a page contains zero or more lines and points, lines connect two points
Actions on objects:
  Page: clear
  Points: Create, delete, move
  Lines: Create delete, move
Attributes of objects:
  Line: Color, style, weight
  Point: Marker type
Actions on attributes:
  Line: Change color, style, weight
  Point: Change marker type

Conceptual design example: a text editor
Objects: files, lines, words
Relations: file is a sequence of lines, lines are a sequence of words
Actions on objects:
  Files: create, delete
  Lines: create, delete, move, copy
  Words: create, delete, move, copy
Attributes of objects:
  Files: size
  Lines: length
  Words: length

Typical elements of conceptual designs
Actions on objects (may also modify relations between objects)
- Create X
- Delete X
- Make X “current”

Attributes of objects
- size
- angle
- position
- linestyle
- font
- cost
- weight
- nationality

Actions on attributes
- set
- modify
- inquire

Relations between objects
- X is a set of Y
- X is a sequence of Y
- X is a triplet (A,B,C)
- X is aligned with Y

Actions on relations between objects
- Remove X from group (or set)
- Insert Y into group (or set)
- Align X with Y
- Remove relation of X aligned with Y

Windowing versus Scrolling Experiment (1982)

Windowing up:

Moves the window up on a fixed world, revealing information which was obscured at the top of the screen

Scrolling up:

Moves the world up under a fixed window, revealing information which was obscured at the bottom of the screen

281 high school students were asked to “move” the view of an alphanumeric display, such that the desired character would be displayed.
The subjects’ first attempt moved the image into view, and thus determines whether scrolling or windowing would be used.

Results:
79% of subjects assumed windowing definition

Is it easier for a user to learn
1. any high-level commands, or
2. a few low-level commands, plus many procedures to map the low-level commands into their equivalent high-level tasks?

Is it faster to use one high-level command, or a sequence of low-level commands?

High-level commands
- tend to be numerous
- slow to learn
- powerful to use
- ease of expression
- less flexible

Low-level commands
- fewer in number
- easier to learn
- less powerful
- combine to obtain power
- more flexible

23 Ways to Draw a Line
1. Between two indicated screen positions
2. Between two indicated points
3. Through a point, horizontal, vertical or both
4. At an angle to a line at an offset distance
5. Parallel to a line at an offset distance
6. Parallel to a line through a point
7. Perpendicular to an entity, near the selected position, from a point
8. Tangent to two selected curves
9. Through a point and tangent to a curve
10. Normal or tangent to a curve through a point on the curve
11. Along the intersection of two planes
12. Perpendicular to a line and tangent to a curve at the selected position
13. At a horizontal or vertical displacement from a point/origin
14. Projected onto a plane, normal projection from the work view
15. Projected normal onto a plane

16. Parallel to a line and tangent to a curve at the selected position
17. Keying in the end points
18. Dividing a line into any number of equal parts
19. The rays when dividing the angle between two lines into any number of equal angles
20. Definition of the conic axes as lines
21. Through a point at an angle to the horizontal
22. Normal to an entity through the selected position
23. Tangent to an entity through the selected position

Large semantic units: each method could be an action, or
Small semantic units: more primitive commands (normal, parallel, tangent, distance, etc.) can be combined to implement methods.
Extensibility is the key.

Evaluation of command design
How well does the user's conceptual model map onto the commands and command sequences?
High frequency tasks are most critical
Must consider task sequences (scenarios)
Difficult to do if user requirement definition:
    is incomplete
    may change over time
    See *Software Architectures: Divine Plan or Digital Darwinism?* (Lewis, 1996)

Semantic Design

Completely design units of meaning between user and computer, but <u>not</u> the form
From user to computer:
    Detailed definition of commands for operating on objects, relations between objects, and attributes of objects
From computer to user:
    Selection of what information needs to be presented to the user

Identify problems that can occur and engineer them out when possible.

A Typical Semantic Specification

Command: Add symbol instance
Information Required: Symbol identifier
Symbol position
Description: An instance of the symbol is created and added to the figure at the designated position. The instance becomes the currently selected object, so that succeeding operations apply to it. The previous currently selected object is no longer selected.
Feedback: The instance is seen on the display, and is highlighted because it is the currently selected object. The previous currently selected object is de-highlighted.
Errors: 1. The symbol identifier is unknown (engineered out by use of menu for selected symbol)
2. The symbol position is outside the viewport (engineered out by constraining positioning device to viewport)

Semantic Design Issues

Structure semantics to ease of learning
- Simple “starter kit” of commands
- 20/80 rule [20% of the commands do 80% of the work]

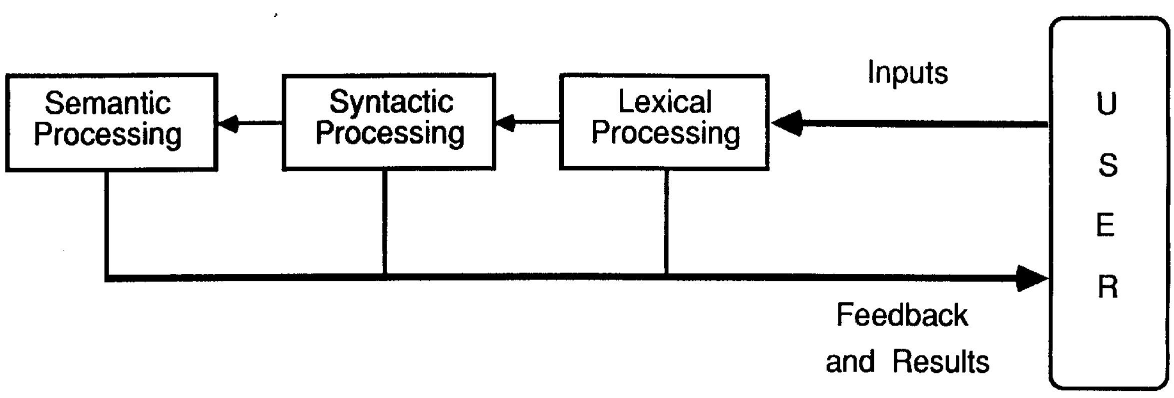
Follow the “principle of least astonishment”
Use defaults
Avoid side effects
Consider amount of information presented to user
- Don’t overwhelm
- Context information is crucial

Feedback of selected commands and object is essential
Selection is separated from operations and is linked only by the curently selected object

Exercise in Designing Command Names for an Electronic Mail System

Function	Command
Begin writing text/mail	
Put more at end of current text	
Produce hard copy	
Preserve text in a file	
Get rid of some words	
Get rid of whole document	
Ship text as mail to addressee	
Terminate session	

Table 2:



Physical design

[From Foley, vanDam, et al. 1987]

Visual Clarity: The meaning of an image should be readily apparent to the viewer. (See Figure 29.)

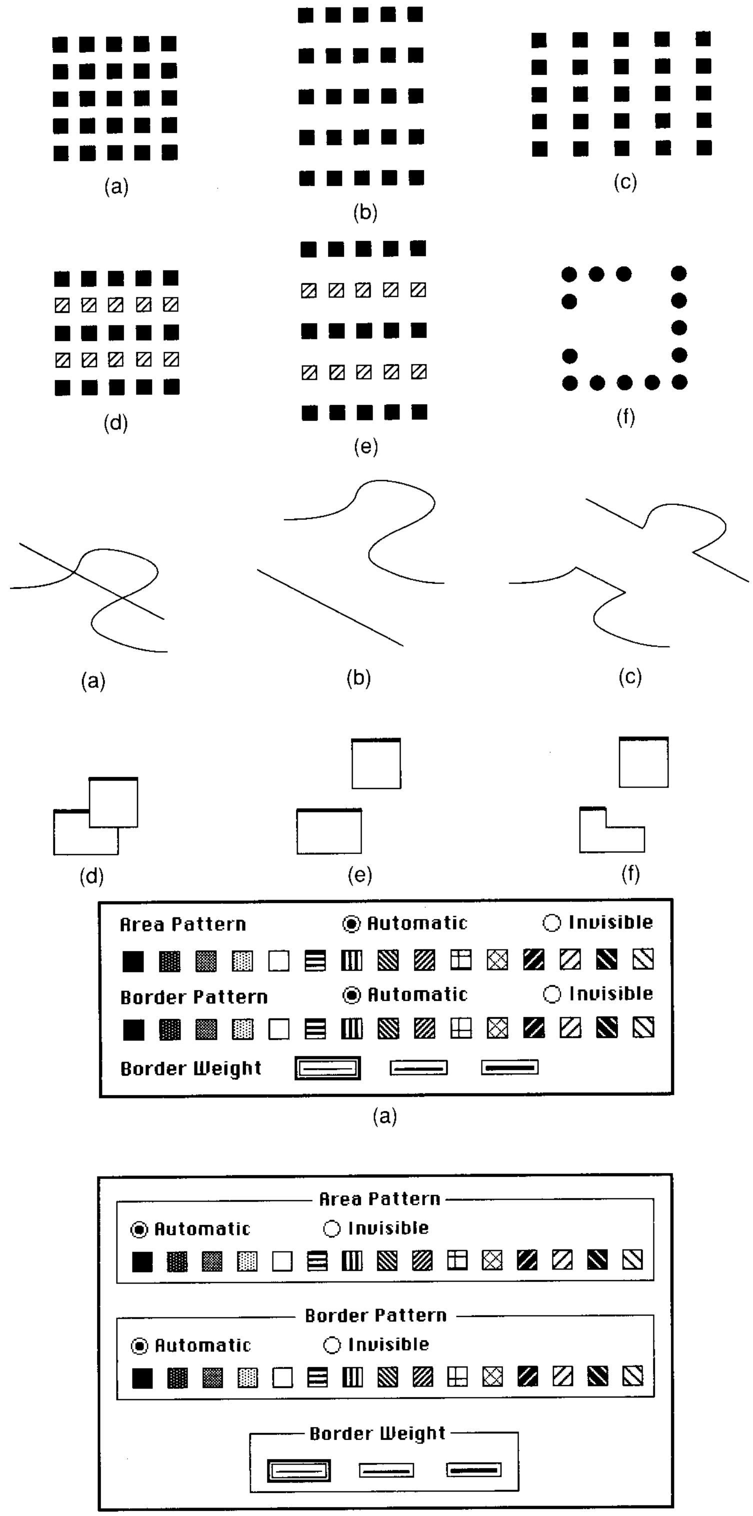


Figure 29.

Visual Codings: Visual distinction between several types of objects. (See Figure 30.)

- Color
- Shape
- Size
- Length...

Not all techniques are capable of coding the same information.

- Nominative: name each different object
- Ordinal: greater than or lesser than relationship
- Ratio: a metric value

Not all codings are equally recognizable.
In order from most recognizable to least recognizable:

1. Position along a common scale
2. Position on identical, nonaligned scales
3. Length
4. Angle between two lines, and line slope
5. Area
6. Volume, density, and color saturation
7. Color hue

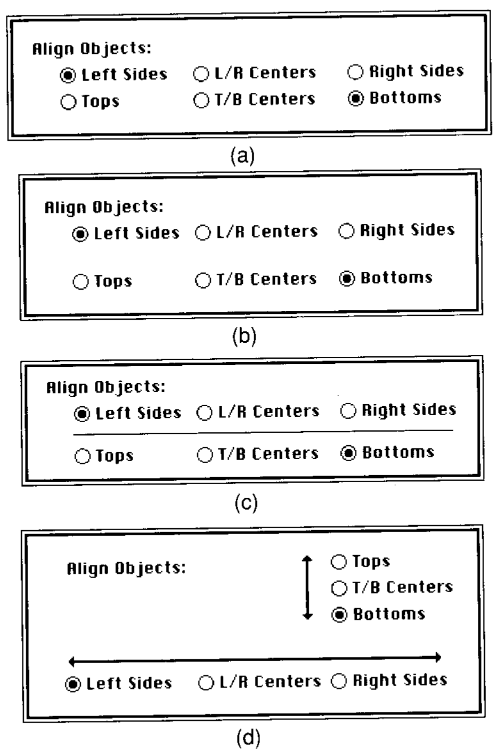


Figure 30.

Visual Consistency: Consistent application of visual-organizational rules and codings, and the combination of visual elements into higher-level graphic objects and icons. (See Figure 31.)

Layout Principles:
Balance (See Figure 32.)

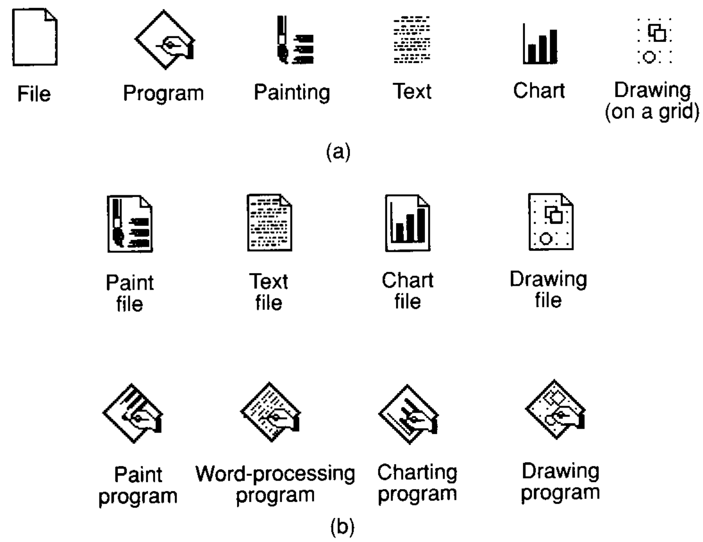


Figure 31.

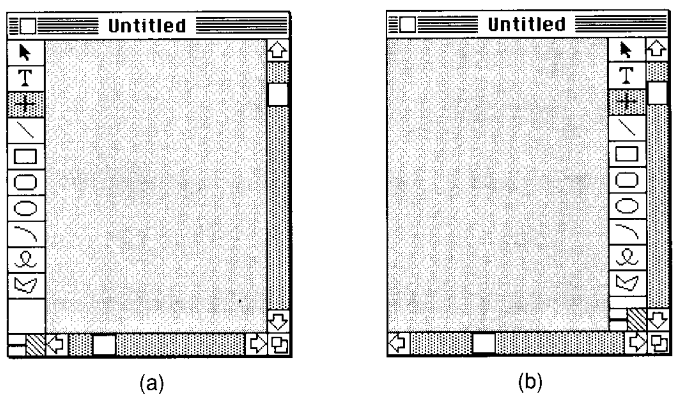


Figure 32.

Gridding (See Figure 33.)

Proportion
Use proper proportion of areas: 1:1 (square), 1:1.414 (square root), 1:1.618 (golden rectangle), 1:2 (double square)

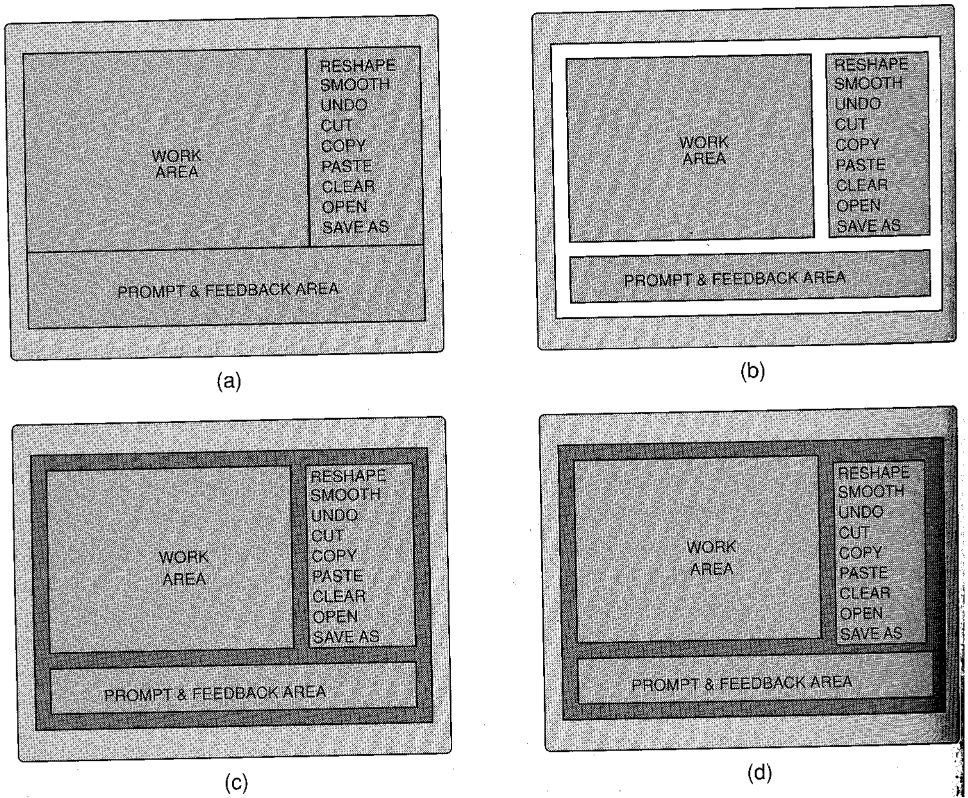

RESHAPE
SMOOTH
UNDO
CUT
COPY
PASTE
CLEAR
OPEN
SAVE AS
WORK AREA
PROMPT & FEEDBACK AREA
(a)
(b)
(c)
(d)

Figure 33.